A Flight/Ground/Test Event Logging Facility

Daniel Dvorak
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
phone: 818-393-1986
email: daniel.dvorak@jpl.nasa.gov

Problem Description

Domain Description

The onboard control software for spacecraft such as Mars Pathfinder and Cassini is composed of many subsystems including executive control, navigation, attitude control, imaging, data management, and telecommunications. The software in all of these subsystems needs to be *instrumented* for several purposes: to report required telemetry data, to report warning and error events, to verify internal behavior during system testing, and to provide ground operators with detailed data when investigating in-flight anomalies. Events can range in importance from purely informational events to major errors. It is desirable to provide a uniform mechanism for reporting such events and controlling their subsequent processing.

Since radiation-hardened flight processors are several years behind the speed and memory of their commercial cousins, and since most subsystems require real-time control, and since downlink rates to earth can be very low from deep space, there are limits to how much of the data can be saved and transmitted. Some kinds of events are more important than others and should therefore be preferentially retained when memory is low. Some faults can cause an event to recur at a high rate, but this must not be allowed to consume the memory pool. Some event occurrences may be of low importance when reported but suddenly become more important when a subsequent error event gets reported. Some events may be so low-level that they need not be saved and reported unless specifically requested by ground operators.

The Desired Program

Your task is to design an object-oriented event logging facility (ELF) that spacecraft programmers will use to instrument flight code, that ground operators will control during mission operations to select different levels of visibility, and that system test tools will connect to in order to audit test results. Every event will be signaled with an associated time stamp, event identifier, and event severity. Some types of events may include additional event-specific data. Any single event may or may not be recorded for subsequent downlink to earth based on an "entry policy" that is sensitive to event type, severity, ID, frequency, and available memory. The tunable parameters of all policies (defined in the next section) will be controlled via ground commands. The interface for signaling the occurrence of an event must incur minimal overhead when the entry policy is set to discard the event. All event data must be handled in a typesafe manner.

You may assume that a "data transport" subsystem exists for uplinking commands from ground to spacecraft and for downlinking data from spacecraft to ground. However, understand that earth-spacecraft communication may be infrequent (once a week), slow (tens of bits per second), and non interactive (round-trip delay from earth to Saturn is about N hours).

You may also assume that a "data management" subsystem exists for saving "data products" such as events and making them available to other subsystems (such as data transport).

Detailed Requirements & Definitions

- **Definition**. An *event* is any noteworthy state, as determined by a system engineer or designer or developer. For example, a bus voltage below 22 volts or a memory pool over 98% full might be considered noteworthy states.
- **Definition**. An event is said to have *occurred* (in a software sense) when it is detected in a conditional statement and can

therefore be acted upon.

- **Definition**. An event occurrence is said to have been *signaled* to Elf when an appropriate Elf signaling function has been called.
- **Definition**. A signaled event is said to have been *logged* if an event record is created, submitted to Data Management, and accepted.
- **Definition**. A *data product* is a data structure to be stored and transported by Data Management. ("Data Management" is the name of a subsystem.)
- **Definition**. An *event object* is an data product containing information describing the occurrence of a particular event.
- **Definition**. An *event type* or *event class* is a data type that specifies the kinds of data that describe an event occurrence.
- **Requirement**. All event object must contain a time stamp, event identifier, and event severity. An event object may contain additional user-defined information as an instance of a user-defined event type.
- **Definition**. An *event identifier* is a label for a kind of event. (An event identifier is useful in distinguishing among different kinds of events that use the same event type.)
- **Definition**. An *event severity* is a measure of the level of importance of an event occurrence.
- **Requirement**. The contents of an event object must be structured and strongly typed so that downstream processing can access the contents in a type safe manner.
- **Requirement**. Elf must define a signaling interface whereby an event occurrence is signaled with the information needed to construct an event object.
- **Requirement**. Elf must define interfaces for controlling entry policy, retention policy, and reporting policy. It must be possible to change the tunable parameters of a policy at run-time without recompilation.
- **Definition**. An *entry policy* controls what signaled events are logged. As an example, a policy may control entry based on event type, event severity, event identifier, frequency of event occurrences, and equality to the previous event.
- **Definition**. A *retention policy* controls how long a logged event is retained. As an example, a policy might depend on factors such as age and number of currently retained events.
- **Definition**. A *reporting policy* specifies how event objects may be grouped into a report. For example, the logging of a severity "red" event of type X could be a trigger to report the last five minutes of events of types Y and Z.
- **Requirement.** A user who defines a new event class must have the option to use a default system-defined policy or to define a class-specific policy. This requirement applies to entry policy, retention policy, and reporting policy.
- **Requirement**. It must be possible to change an entry policy at runtime, i.e., no source code changes and no recompilations.
- **Requirement**. For reasons of runtime efficiency in high-performance applications, at least one of the signaling interfaces must be designed for speed in ignoring disabled events.
- **Requirement**. Elf must support at least three levels of event severity: a "green" level for purely informational events, a "yellow" level for warnings, and a "red" level for errors. (The names "red", "yellow", and "green" are merely suggestive, not required.)
- **Requirement**. It should be possible to include with a logged event the source location where it was signaled. This helps distinguish between events that otherwise have the same identifier, same type, and same severity.
- **Desideratum**. Since brevity is a virtue to most programmers, Elf should provide at least one signaling interface for basic events that can be written in a compact form. The intent of this requirement is to make it easy to instrument an application's source code, particularly during early design and debugging.

Non Requirements

1. **Non requirement**. There is no limit on the number of event types that may be defined.
2. **Non requirement**. There is no requirement for a signaling interface that can be conditionally compiled down to zero run-time overhead, i.e., compiled out of existence.
3. **Non requirement**. The preceding requirements deliberately do not prescribe whether and how exceptions might be used to signal error events. Users who wish to use exceptions may do so, and could (for example) define an error event hierarchy derived from the base event class so that a catcher could specify how broad of a class of error events it can handle. Naturally, each subsystem can define its own error hierarchy, independent of other subsystems, so there's no need for a standard hierarchy. Likewise, the choice of whether or not to use exceptions can be made on a subsystem basis.

4. **Non requirement**. There is no requirement for Elf to maintain statistics such as the number of times that an event condition has been *checked*.
   Non requirement. There is no requirement for Elf to provide a way to force the occurrence of an event, such as for testing purposes.

Use Cases

1. A programmer defines an application-specific event class.
2. A programmer instruments source code to signal an event if it occurs.
3. A running application program signals occurrence of an event.
4. Data management accepts an event object for logging.
5. An Elf signaling mechanism checks the appropriate entry policy.
6. A ground operator adjusts the tunable parameters of a specific policy.

References for Further Study

- I'll try to find a published article on the Mars Pathfinder event reporting facility (but it was based on macros for a non-OO C environment).
- (I welcome suggestions for other relevant articles.)

If you have comments or suggestions, email me at *daniel.dvorak@jpl.nasa.gov*